情報処理学会論文誌　Vol. 49　No. 12　4067–4079 (Dec. 2008)

複数のGPUを用いた粒子法シミュレーションの並列化

原　田　隆　宏[†1]　政　家　一　誠[†2,†3]
越　塚　誠　一[†2]　河口　洋一郎[†2]

本論文では，粒子法シミュレーションを複数のプロセッサを上で並列化する方法を提案する．計算を分割する際にすべてのプロセッサの計算を管理するプロセッサを配置するサーバクライアント型の計算モデルでは，そのプロセッサの処理が並列化できないため並列化においてボトルネックとなりうる．そこで本研究ではそのような全体を管理するプロセッサを置かず，すべての計算プロセッサがそれぞれのデータを管理するピアツーピア型の計算モデルを提案する．また粒子法シミュレーションでは計算要素である粒子が計算領域内を自由に動くため，それぞれのプロセッサに送るデータを動的に生成しなければならない．そこでこの並列化のオーバヘッドをおさえるため，近傍粒子探索を効率化するために構築した格子のデータを用いるデータ管理方法を提案する．そして本手法を複数の Graphics Processing Unit（GPU）を用いて実装し，Distinct Element Method（DEM）のシミュレーションを行い計算速度を測定し，本手法の有用性を示す．

Parallelizing Particle-based Simulation on Multiple GPUs

Takahiro Harada,[†1] Issei Masaie,[†2,†3]
Seiichi Koshizuka[†2] and Yoichiro Kawaguchi[†2]

This paper presents a method to parallelize particle-based simulations on multiple processors. The computation model that puts a processor managing computations of all the processors is not ideal for parallelization because the managing processor can be a bottleneck of the simulation because it cannot be parallelized. Therefore, we propose another strategy wherein all the processors manage their own data, respectively. When parallelizing a particle-based simulation, the transfered data between processors have to be calculated in each time step because particles moves freely in the computational domain and this can be an overhead of parallelization. In order to avoid additional computation, we used the grid constructed to improve the efficency of neighboring particle search to compute the data have to be send. The present method was used to implement Distinct Element Method (DEM) on multiple Graphics Processing Units (GPUs) and the computational times are measured.

1. 序　　論

粒子法シミュレーションはコンピュータグラフィックスの分野で重要な技術の 1 つであり，1 枚の画像を作るのに長い時間をかける映像制作からリアルタイムアプリケーションまで幅広く用いられている[21]．これらの分野ではシミュレーションが高速なほど良い．特にリアルタイムアプリケーションでは計算の速度が実時間程度である必要があり，シミュレーションの速度への要求が厳しい．そこでシミュレーションを高速化するために，マルチコアのストリームプロセッサの 1 つである Graphics Processing Unit（GPU）を用いた研究も行われてきた[13]．代表的なストリームプロセッサとしては GPU，Cell Broadband Engine が存在するが，シーケンシャルに処理を行う CPU も近年ではマルチコアになってきており，今後はこのようなマルチコアプロセッサの性能を引き出すことができるアルゴリズムの重要性が高まっていく．しかし効率を上げるためにコア数を増加させるのも限界があると考えられる．そうなると複数のストリームプロセッサをさらに並列に配置して，さらに計算効率を上げる必要性が出てくると考えられる．このような計算環境ではストリームプロセッサで実行できるアルゴリズムを開発するだけでなく，それらが並列で計算するため，もう 1 段階の計算の並列化を考えなければならない．

本研究では粒子法シミュレーションに対して複数の GPU を用いて 2 段階の並列化を行うことのできる手法を開発する．このような 2 段階の並列化を考えるときには 2 種類の計算モデルが考えられる．1 つはサーバクライアント型の計算モデルであり，サーバとなる全体の計算を管理するプロセッサを置き，そのプロセッサが他のプロセッサに計算を割り当て，その結果を回収する．しかしこの計算モデルではサーバとなるプロセッサの処理は並列化されていないため，クライアントのプロセッサを増やしても効率化はできない．そのため，サーバとなるプロセッサの処理がボトルネックとなりうる．またサーバはクライアントに計算データを渡し，計算結果を受け取るため，多量のデータ転送が必要になり，並列化のオーバヘッドが大きくなる．そこで本研究ではサーバとなるプロセッサを置かず，すべての計算

†1 Havok
†2 東京大学
The University of Tokyo
†3 プロメテックソフトウェア株式会社
Prometech Software Inc.

© 2008 Information Processing Society of Japan

プロセッサがそれぞれのデータを管理するピアツーピア型の計算モデルを提案する．この計算モデルでは全体のデータをすべて管理するプロセッサを置かず，それぞれのプロセッサが計算データを分散して管理する．計算はそれぞれのプロセッサが持つデータのみでは行えないため，他のプロセッサが持つデータを受け取る必要がある．この転送するデータの管理もそれぞれのプロセッサが並列で処理を行うため，計算全体ではサーバクライアント型の計算モデルのように並列化できない部分がなく，並列化のオーバヘッドを減らすことができる．

粒子法シミュレーションを複数のプロセッサ上で並列化するときには，もう 1 つ考慮しなくてはならないものがある．計算要素間の接続が固定された格子のシミュレーションでは，計算領域を分割し 1 つのプロセッサが 1 つの計算領域の中にある要素の計算を行う．そして隣接する領域を計算しているプロセッサから自分の計算要素に隣接している要素の物理量を受け取り，自分の計算要素の値を求める．計算要素間の接続が固定されているため，計算領域の分割は計算の前に 1 度行っておけばよく，データを転送する必要がある要素も固定されているため，このリストも計算の前に作成することができる．しかし粒子法シミュレーションではそのようにすることができない．粒子法では粒子が自由に動くため，計算領域を分割してもその中に存在する粒子は固定されていない．すなわち，計算領域を分割してプロセッサに割り当てるときには，毎タイムステップにおいて計算領域内に存在する粒子を求める必要がある．さらに計算要素間，つまり粒子間の接続が固定されておらず，毎タイムステップごとに接続情報を求めなければならないため，計算を領域で分割したとしても，隣接する領域を計算しているプロセッサに転送するデータも動的に変化する．このように粒子法シミュレーションを複数のプロセッサ上で並列化するときには，並列化のオーバヘッドが大きくなってしまう可能性があり，これらをどのように処理するかが重要になってくる．本研究では粒子法シミュレーションの計算を領域で分割し，隣接した領域を計算しているプロセッサに転送するデータの管理には，近傍粒子探索を効率化するために構築した格子データを再利用することによって並列化のオーバヘッドをおさえる手法を提案し，GPU 上で Compute Unified Device Architecture（CUDA）を用いた実装を行う[12]．

また複数のプロセッサでリアルタイムシミュレーションを行い，レンダリングまで行う場合は，シミュレーションの負荷は複数のプロセッサで分散するのに対し，レンダリングを行う GPU はすべてのプロセッサからのデータを受け取り，処理を行わなければならない．そこでレンダリングを行う GPU の負荷を軽減するため，シミュレーションを行っている複数の GPU でデータの縮小を行い，レンダリングの負荷を分散させると同時に，システム全体のデータ転送量を減少させる手法も提案する．

2. 関連研究

粒子法では計算要素が固定の接続情報を持たない粒子であり，粉体の計算手法である Distinct Element Method（DEM）や[1]，流体の計算手法である Smoothed Particle Hydrodynamics（SPH）[10] や Moving Particle Semi-implicit（MPS）Method が研究されてきた[9),19),20]．しかしこの固定の接続情報を持たないという点は粒子法の利点でもあり，欠点でもある．つまり毎タイムステップにおいて近傍粒子粒子を探索し，接続情報を計算しなければならないため，計算コストが高くなってしまう．そこで GPU をストリームプロセッサとして用い，計算を高速化する研究も行われてきた．DEM，SPH，粒子法を用いた剛体のシミュレーション，流体剛体連成まで様々な粒子法を並列化し，GPU で計算するアルゴリズムが開発され，高速化されてきた[3),5),6),22]．

複数のプロセッサを用いて計算を並列化する研究も行われてきた．序論で議論したように計算要素間の接続関係が固定されている問題は比較的容易に複数のプロセッサを用いて並列化することが可能である．たとえば Thomaszewski らによる布の計算や Ribeiro らによる有限要素法の計算があげられる[15),17]．またすべての計算を陽解法で行う Lattice Boltzmann Method（LBM）は特に複数のプロセッサを用いた並列化に向いており，複数の GPU を用いた研究も行われている[2]．

しかし粒子法のシミュレーションでは計算要素間の接続関係が固定されていないため，格子を用いるシミュレーションの並列化手法をそのまま適用することはできない．粒子法を複数のプロセッサを用いて並列化した研究としては GRAvity PipE（GRAPE）という専用ハードウェアを用いた研究があるが，この研究では SPH のすべての計算を行うことができないため，CPU がサーバとなり GRAPE 上の計算を管理しているため，これらの間での多量のデータ転送が必要である[11]．ここで用いられている計算モデルがサーバクライアントモデルである．この研究とは異なり，より効率の良い計算モデルを採用している研究もある．たとえば粒子法の 1 つである Macro-scale Pseudo-Particle Modeling（MaPPM）を並列化した研究がある[18]．また近年では，Stratford らは LBM による流体のシミュレーションを並列化するだけではなく，さらに粒子の計算も並列化した[16]．しかし，これらの研究では複数のプロセッサがいくつかの粒子のデータを重複して保持するため，数回のデータ転送が 1 タイムステップの計算において必要であるという欠点があった．またこれらの研究では粒子の物理量を更新した後にデータをプロセッサ間で交換するが，この処理をどのように行うかということについては考察がない．これは 1 タイムステップの計算時間が数

© 2008 Information Processing Society of Japan

秒以上と長かったため，データの選別にかかる時間が全体の計算時間に占める割合が小さいため，この処理はほとんど問題にならなかったからだと思われる．MPS 法を PC クラスタを用いて並列化し，高速化した研究も存在する．入部らは MPS の連立方程式の計算を並列化して高速化を行った[23]．しかし連立方程式の計算以外の部分が並列化されていないため，それ以外の部分の計算負荷の割合が高い手法に適用した場合には並列化効率が上がらない．よってこの並列化は MPS のように連立方程式を解く必要のある手法には有効であるが，DEM や SPH など連立方程式を解く必要のない手法には適用することができない．

このように様々な研究者が複数のプロセッサを用いた並列化を研究してきたが，リアルタイムアプリケーションに複数のプロセッサを用いた並列化を適用した例は我々の知る限り存在しない．しかし我々の研究目的はリアルタイムアプリケーションに複数のプロセッサを用いて並列化することであるため，既存研究では大きな問題にならないようなことも問題になってくる．たとえば粒子法のシミュレーションでは転送する必要のあるデータの選別などである．

本研究の新規性は具体的には以下のようにまとめられる．粒子法のシミュレーションを並列化する際に，計算粒子を重複せずに管理する手法を開発した．これは 1 粒子が複数のプロセッサで計算されることがないということである．これによって計算粒子を重複して管理していた場合に比べてデータ転送回数が減り，並列化の効率を良くすることが可能になる．そしてこの計算に必要なゴースト領域の導入方法も既存の粒子法の並列化を行った論文とは異なる．また本論文では，近傍粒子探索を効率化するために計算した格子のデータを再利用するプロセッサ間のデータ転送方法を提案する．データ転送方法については前述のように既存研究では焦点が当てられてこなかった．本研究では複数の GPU を用いてリアルタイムアプリケーションを並列化する手法を開発したが，リアルタイムアプリケーションで複数のプロセッサを用いて並列化を行った研究はなかった．上記の本研究の新規性は 1 タイムステップの計算時間がきわめて短いリアルタイムアプリケーションを並列化する際に直面する問題点の解決手法であるとまとめられる．

3. 格子を用いた近傍粒子探索の効率化

粒子法では近傍粒子探索を毎タイムステップ行う必要があり，これを効率化するために，格子を導入することができる．本手法は粒子法シミュレーションにおいて格子が導入されていることを前提とするため，本章で格子を用いた近傍粒子探索の効率化について簡潔に述べる．なお詳細は参考文献 4)，21) に譲る．

ある粒子の物理量の計算にはその粒子の近傍に存在する粒子の物理量が必要になる．近傍粒子を全粒子から探索したのでは粒子数を n とすると $O(n^2)$ の計算になり，粒子数の増加にともない計算コストが大きくなってしまう．そこで計算領域を覆う格子を用意し，その格子のそれぞれのボクセル内の領域に存在している粒子番号を格納する．この粒子番号が格納された格子を用いることによってある粒子の近傍粒子はその粒子が存在している格子の周囲の格子に粒子番号が格納されている粒子に限定され，計算コストは $O(n)$ に改善される．ボクセルの大きさの選択は任意であるが，本研究で用いる DEM の場合は衝突粒子を探索するため，ボクセルの 1 辺の大きさを粒子の直径 l_0 とすることで，衝突粒子はその粒子が格納されているボクセルの周囲の 3^3 個のボクセル内に存在する粒子に限定されるようになり，最も効率的である．

4. 計算の分割

複数のプロセッサで 1 個のシミュレーションを行うときには，計算を分割しなければならない．計算の分割の方法としてはいくつかの選択肢があるが，本研究では計算領域を分割し，それぞれのプロセッサが割り当てられた領域内に存在する粒子の物理量を計算するようにし，割り当てられた領域内に存在する粒子の物理量のみを更新する．

粒子法ではある粒子の物理量を計算するときに近傍粒子の物理量が必要になる．DEM では近傍粒子と衝突計算を行い，SPH では近傍粒子の持つ物理量を積分する．しかしあるプロセッサが計算している領域の境界付近に位置する粒子の近傍粒子は，そのプロセッサの計算領域の外に存在し，隣のプロセッサの計算領域に存在していることがある．このような場合には隣接するプロセッサのメモリに存在するその近傍粒子のデータにアクセスしなければならない．この処理を境界に位置するすべての粒子において必要に応じて行うのは小さいデータを複数回転送しなければならず非効率的であるため，本研究ではゴースト領域を導入する．ここで計算領域が

$$C = \{x | s < x \leq e\} \tag{1}$$

であり，2 つのプロセッサ p0，p1 が 1 つの計算を行っているとする．計算領域を X 軸に垂直な平面で分割して，それぞれの計算領域を

$$C_0 = \{x | s < x \leq m\} \tag{2}$$

$$C_1 = \{x | m < x \leq e\} \tag{3}$$

とする．ここで $m = (s + e)/2$ であり計算領域の中点の X 座標である．すると p0 のゴースト領域 $G_{1\rightarrow 0}$ は C_0 に隣接した幅 g の C_1 内の領域であり，

© 2008 Information Processing Society of Japan

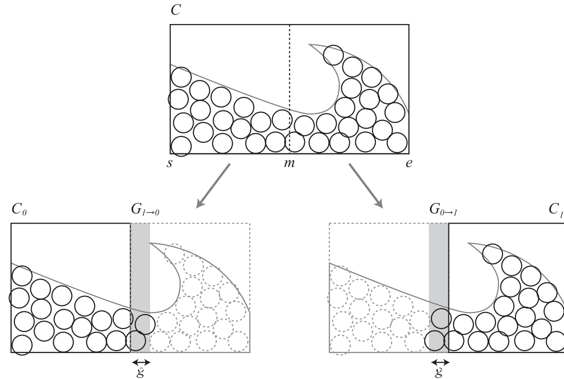

図 1　2 個のプロセッサを用いたときの計算の分割
Fig. 1　Decomposition of the computation on 2 processors.

$$G_{1\to 0} = \{x | m < x \leq m + g\} \tag{4}$$

とし，また p1 のゴースト領域 $G_{0\to 1}$ は同様に C_1 に隣接した C_0 の領域であり

$$G_{0\to 1} = \{x | m - g < x \leq m\} \tag{5}$$

と定義する．n 個のプロセッサを用いる場合は，計算領域を n 分割し，0 番目と $n-1$ 番目のプロセッサ以外は 2 個のプロセッサの計算領域に隣接しているため，2 個のゴースト領域を持つようになる．粒子の影響半径を $r_e = g$ とすると，プロセッサ p0 が計算している粒子の近傍粒子は $C_0 \cup G_{1\to 0}$ の領域内に存在していることになり，計算の前にゴースト領域の粒子データを受け取れば近傍粒子探索のときに隣接するプロセッサのデータを参照しなくてもよくなる．このゴースト領域内の粒子をゴースト粒子と呼ぶ．p0 はこのゴースト粒子の物理量は更新せず，C_0 の領域内に存在する粒子の物理量のみ更新する．いずれの粒子も $C_0 \cup C_1$ に存在しているため，すべてのプロセッサが，割り当てられた領域内の全粒子の物理量を更新すれば，すべての粒子の物理量が更新される．すなわち $G_{1\to 0} \subset C_1$ であり，$G_{0\to 1} \subset C_0$ であるため，各プロセッサはゴースト粒子の物理量を更新しなくても問題はない．**図 1** に 2 個のプロセッサを用いた場合のこれらの領域の関係を示す．

5. データの管理

粒子法では格子法とは異なり計算粒子は自由に動くことができるので，計算中に粒子が他のプロセッサの計算領域に出ていくこともあり，また逆に他のプロセッサの領域から入ってくることもある．粒子が動くということはゴースト粒子も各タイムステップにおいて変化するということである．よって計算のデータを管理する必要性が生じる．最も容易なデータの管理方法は，サーバ，クライアントモデル的な管理方法であり，サーバとなるプロセッサを用意して，そのプロセッサが全粒子の物理量を保持する．そして各タイムステップごとにクライアントに必要な情報を送る方法である．しかしこの方法では毎タイムステップにおいてサーバはすべてのクライアントから全粒子のデータを受け取り，さらに全クライアントに送るデータを計算しなければならず，毎タイムステップにおいて多量のデータ転送を行う必要がある．さらに処理をサーバが行っているときにはクライアントは処理を行うことができないため，計算のボトルネックとなりうる．そこで本研究ではサーバを置かずに，各プロセッサがデータを管理するようにした．本章ではデータ管理方法を述べる．

5.1　重複のない計算粒子の管理法

簡単なデータ管理方法はすべてのプロセッサがすべての粒子データを持つという方法である．そしてそれぞれのプロセッサに割り当てられた領域内に存在する粒子の物理量の更新を行う．しかしこの手法ではすべての GPU がすべての粒子データのメモリを確保しなければならず，メモリ効率が悪い．そこで本研究ではそれぞれの GPU がそれぞれの計算領域内に存在する粒子のデータのみをメモリに持ち，計算を行う．

5.1.1　データの送信

前述のようにあるプロセッサの計算領域の境界付近の計算を行うためには隣接するプロセッサの持つデータが必要である．また粒子が計算領域にとらわれず自由に動くため，あるプロセッサの計算領域から出た粒子のデータを隣接するプロセッサに渡す必要がある．よって 1 タイムステップごとにそのプロセッサの計算領域から出ていく粒子と隣接したプロセッサの計算に必要なゴースト粒子を送る必要がある．ある時刻 t で p0 が計算している粒子 i の座標を x_i^t として，そのタイムステップで計算した後の時刻 $t+\Delta t$ での粒子の座標を $x_i^{t+\Delta t}$ とする．粒子 i は p0 の計算領域内に存在するため，$x_i^t \in C_0$ である．まずこの計算領域 C_0 から C_1 に出ていく粒子は

$$EP_{0\to 1}^{t+\Delta t} = \{i | m < x_i^{t+\Delta t}, x_i^t \leq m\} \tag{6}$$

の粒子である．また p0 の計算領域内に存在する p1 に必要なゴースト粒子は

$$GP_{0\to 1}^{t+\Delta t} = \{i | m - g < x_i^{t+\Delta t} \leq m\} \tag{7}$$

に存在する粒子である．よって隣の p1 に送らなければならない粒子は，式 (6) と式 (7) より，

$$\begin{aligned} SP_{0\to 1}^{t+\Delta t} &= EP_{0\to 1}^{t+\Delta t} + GP_{0\to 1}^{t+\Delta t} \\ &= \{i | x_i^{t+\Delta t} > m - g, x_i^t \leq m\} \end{aligned} \tag{8}$$

© 2008 Information Processing Society of Japan

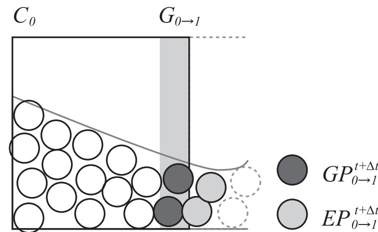

図 2　プロセッサ 1 からプロセッサ 2 へ送る粒子
Fig. 2　Particles send from processor 1 to processor 2.

Input array	1	0	3	1	2	0	0	1	2	1
Output array	0	1	1	4	5	7	7	7	8	10

図 3　Prefix sum
Fig. 3　Prefix sum.

である（**図 2**）．

データの送信方法にもいくつかの選択肢がある．1 つ目の方法はすべてのデータを送り，送り元のプロセッサの領域に存在した粒子のデータを，受け取った先のデータに上書きすることである．しかし実際は隣接したプロセッサの持つ全粒子のデータは必要ではなく，計算領域に入ってくる粒子とゴースト粒子の物理量のみ受け取り更新すればよい．また一般的に送る必要のあるデータ量が全粒子の物理量の中に占める割合は少ないため，この送信方法は非効率的である．最も効率が良いのは $SP_{0\to 1}^{t+\Delta t}$ の粒子の集合のみ送ることであるが，それには全粒子からこの条件に合致した粒子のみを選択しなければならない．そのためには全粒子の中でその条件にあった粒子にフラグを立て，そのフラグの全粒子の中での番号を計算して，データを送るバッファに書き込まなければならない．このように送るデータを選択するには Prefix sum を用いることができるが，CUDA を用いた実装ではまず有効な粒子にフラグを立てるカーネル，Prefix sum を求めるカーネル，有効なデータを Prefix sum で計算したアドレスに移動させるカーネルが必要であり，計算量が増加する[7]．Prefix sum とは**図 3** に示すように入力とする配列 $a_0, a_1, a_2, \cdots, a_{n-1}$ に対して，$b_i = \sum_{j<i} a_j$ の配列 b のことである．

そこで近傍粒子探索を効率化するために構築した格子を用いて他のプロセッサに送るデータを選択する．格子のボクセルに格納された粒子番号を参照することで送信する必要のある粒子を探すことができる．つまりボクセルの X 座標 x_v が $x_v > m - g$ のボクセルに格納されている粒子が隣のプロセッサに送る粒子である．しかし格子に粒子番号を格納するために用いた粒子座標は，タイムステップで更新された $x_i^{t+\Delta t}$ の座標ではなく，更新される前の x_i^t であるため，$x_v > m$ のボクセルを参照したのでは $SP_{0\to 1}^{t+\Delta t}$ の粒子の集合を見つけることはできない．ここで再度格子に粒子番号を格納することもできるが，やはり計算量が増加する．そこで本研究では時刻 t で構築した格子を再利用する．粒子法のシミュレーションでは一般的に任意の大きさのタイムステップを刻めるわけではない．計算安定条件として 1 タイムステップで進む粒子の距離が，粒子の直径以下であるというクーラン条件を課す．つまり粒子の速度，時間刻み幅，粒子の直径をそれぞれ v，Δt，l_0 とすると $v\Delta t/l_0 < c$ と表せる．ここで c はクーラン数と呼ばれる．計算においてクーラン数に $c < 1$ の条件を課すということは，前述のように粒子の 1 タイムステップでの移動は初期粒子間距離以下に制限されるということである．また近傍粒子探索を効率化する格子のボクセルの大きさを粒子の直径と同じにしているため，粒子は 1 タイムステップにおいて 1 ボクセル以上移動することはない．よって $x_i^{t+\Delta t} > m - g$ の粒子は $x_i^t > m - g - l_0$ の条件を満たす．すなわち隣接したプロセッサに送る粒子は $x_v > m - g - l_0$ の条件を満たすボクセルに格納された粒子であるといえる．また粒子の集合 $EP_{0\to 1}^{t+\Delta t}$ は，時刻 t では p0 によって計算されるため，$x \leq m$ である．同様にクーラン数の条件から粒子の集合 $EP_{0\to 1}^{t+\Delta t}$ は $m - l_0 < x^t \leq m$ の条件を満たす．すなわち p0 から p1 に送る粒子 $SP_{0\to 1}^{t+\Delta t}$ は時刻 t において**図 4** に示すように，

$$S'_{0\to 1} = \{m - d - l_0 < x \leq m\} \tag{9}$$

に存在する粒子であり，$d = r_e$ のとき領域 S' は，

$$S'_{0\to 1} = \{m - 2d < x \leq m\} \tag{10}$$

となる．

この選択された粒子の物理量を送るためにデータを格納するバッファを用意しなければならない．均一格子を用いた場合で格子の X 軸での断面が $v_y \times v_z$ であるとき，$g = r_e$ のときには X 軸方向に 2 ボクセル分のデータを送信する必要があり，1 ボクセルに格納される最大粒子数が n であるときには $v_y \times v_z \times 2 \times n$ の粒子のデータを格納できるバッファを用意して，格子を参照して送るデータを作成する．

5.1.2　データの受信

データの受け取りは，すべてのプロセッサが同じ粒子番号を用いている場合には，隣接したプロセッサが送ったデータを受け取り，そのデータ内の粒子番号の粒子の物理量を上書き

© 2008 Information Processing Society of Japan

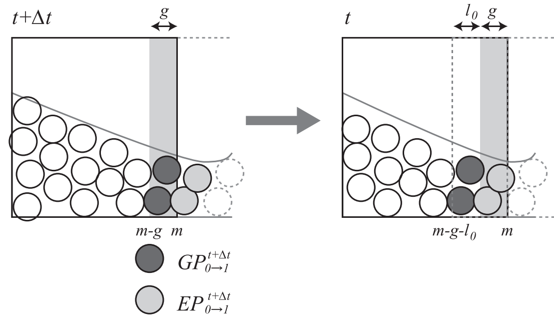

図 4　送信する粒子データ．灰色の領域が p0 の計算領域内のゴースト領域であり，図左において p0 から p1 に送信する粒子を 2 色で示している．これらの粒子の時刻 t での配置を図右に示す

Fig. 4　Particle data sent to the neighboring processor. The gray region is the ghost region in the computation domain of p0 and the colored particles are the particles sent from p0 to p1 in the left figure. The distribution of these particles at time t is shown in the right.

することによって，受け取った粒子の物理量を最新のものに更新すればよく容易である．しかし本手法では，そのようにすることはできない．これはそれぞれのプロセッサが独自の粒子番号を用いて粒子を管理しており，境界に存在するある粒子の番号は，その境界を挟む両方のプロセッサで，それぞれ異なった粒子番号として認識されているからである．そこで受け取ったデータをそのプロセッサの持つデータに追加する．しかしこのときに注意しなければならないのは粒子データの重複である．粒子データの重複を起こさないために，隣接したプロセッサに送るデータを作成する前に，前のタイムステップでゴースト領域に存在した粒子のデータを削除する．粒子を移動させたあとの座標がゴースト領域に存在する粒子を削除するのではないことに注意する必要がある．これは以下の 2 つの命題が真であることから証明することができる．

- ある時刻 t で $G_{1\to0}$ に存在する粒子の集合 $GP^{t}_{1\to0}$ は，次の時刻 $t+\Delta t$ で隣接した計算領域から送られてくる粒子の集合 $SP^{t+\Delta t}_{1\to0}$ に含まれる．
- ある時刻 t で C_0 に存在し，次の時刻で $G_{1\to0}$ に存在する粒子の集合 $EP^{t+\Delta t}_{0\to1}$ は，次の時刻 $t+\Delta t$ で隣接した計算領域から送られてくる粒子の集合 $SP^{t+\Delta t}_{1\to0}$ に含まれない．

まず 1 つ目の命題において，

$$SP^{t+\Delta t}_{1\to0} = \{i | m < x^{t}_{i} \leq m + d + l_0\} \tag{11}$$

の領域に存在する粒子であり，

$$GP^{t}_{1\to0} = \{i | m < x^{t}_{i} \leq m + d\} \tag{12}$$

である．よって式 (11) と式 (12) より，$GP^{t}_{1\to0} \subset SP^{t+\Delta t}_{1\to0}$ である．これにより，時刻 t においてゴースト粒子であったものは，データを受け取る前に削除しなければならないことが分かる．

また 2 つ目の命題において，

$$EP^{t+\Delta t}_{0\to1} = \{i | m - d < x^{t}_{i} \leq m, m < x^{t+\Delta t}\} \tag{13}$$

であり，

$$SP^{t+\Delta t}_{1\to0} = \{i | m < x^{t}_{i} \leq m + d + l_0\} \tag{14}$$

なので，$EP^{t+\Delta t}_{0\to1} \notin SP^{t+\Delta t}_{1\to0}$ である．よって時刻 $t+\Delta t$ においてゴースト領域に存在する粒子は削除してはいけないことが分かる．受け取る粒子とそのプロセッサの計算領域内の粒子は重複がないため，受け取ったデータをそのプロセッサの持つデータの後ろに付け加えるだけでよい．

格子を用いて送るデータを構築する場合は送るデータ配列に粒子が存在しないボクセルが存在する．またすべてのボクセルがボクセルに格納できる最大粒子数の粒子番号を保持しているわけではないため，無効なデータが存在する．そのため，データの送受信が繰り返されると無効なデータが蓄積してしまい，粒子データの配列が拡大し続けてしまう．これを防ぐために粒子のデータをソートする前に（このソートについては後述する）配列の圧縮を行い無効なデータを取り除く．具体的には Prefix sum を用いて配列の圧縮を行う．

5.1.3　アルゴリズム

このようにゴースト粒子を用いて計算領域の境界の計算を行い，格子を利用してデータの送受信をする．単一プロセッサでのアルゴリズムにゴースト粒子の削除とデータの送受信を追加することで複数のプロセッサで計算を行うことができるようになる．GPU 間では直接データの転送を行うことができないため，各 GPU は用意した境界のボクセルの粒子データをビデオメモリからメインメモリに送る．データの受信は隣接したプロセッサがデータの送信を完了した後に行う必要があるので，すべてのプロセッサで同期をとる．後述する計算結果は並列化に OpenMP を用いたため，すべてのスレッドの同期は OpenMP の命令文を挿入することで行った．それからそれぞれが送られたデータをメインメモリからビデオメモリに読み込み，受け取った粒子の物理量を更新する．これが 1 タイムステップの計算の流れである．

6.　格子の選択

5 章で述べた方法でプロセッサ間のデータの転送を行うには，近傍粒子探索を効率化する

© 2008 Information Processing Society of Japan

ために格子を導入していることを前提にした．最も単純な格子は均一格子であるが，均一格子は計算領域全体のメモリを確保する必要があるため，メモリ効率が悪い．そこで本研究ではこの均一格子の問題点を改善するために Harada らによって提案されたスライスグリッドを用いる[4]．

6.1　スライスグリッド

スライスグリッドでは計算領域内の 1 軸をとり計算領域をこの 1 軸に直交するスライスに分割して，それぞれのスライスで軸平行バウンディングボックスを求め，その内部のメモリのみを確保する．これによってメモリ効率が向上し，さらに粒子番号が格納されているボクセルが密に格納されるようになるため，キャッシュ効率も上がり計算速度も向上する．さらに粒子番号と同様に粒子データも粒子の空間配置でソートすることでこれらのデータのキャッシュ効率も向上し計算の高速化が可能である．またプロセッサ間のデータの転送においても均一格子を用いるより，スライスグリッドを用いた方が転送データ量も少なくなる．本研究ではまずスライスグリッドを用いた実装をさらに高速化するために，ソートを導入した．本章ではまずこのソートの導入について述べ，最後にスライスグリッドを用いたときの複数プロセッサ間でのデータの転送について述べる．

6.2　スライスグリッドへのソートの導入

スライスグリッドを用いるとメモリ効率を上げることができるだけでなく，ボクセルのデータが密に格納されるため，キャッシュ効率も向上し，計算も効率的になる．さらに粒子の物理量のメモリ配置が粒子の空間配置になっている場合は，近傍粒子のデータがメモリの近くの領域に連続して存在するため，粒子の物理量へのアクセスにおいてもキャッシュ効率が向上する．しかし粒子法のシミュレーションでは計算要素である粒子の間に固定された接続がなく，自由に動き回ることができるので，計算が進行するにつれてメモリ内の粒子の物理量の配置がランダムになってしまい，キャッシュ効率が下がり計算速度が低下する．よってスライスグリッドのみを用いた場合よりもシミュレーションを高速化する 1 つの選択肢は粒子の物理量を粒子の空間配置通りにソートすることである．しかしソートによって近傍粒子探索は効率化されて高速化されても，ソートのペナルティがそれ以上であったらシミュレーション全体は高速化されない．

挿入ソートはほぼ整列したリストのソートはとても効率的に行うことができるため，フレーム間にコヒレンシのある粒子法シミュレーションのようなほぼ整列したリストに適したソートが有効である．しかし挿入ソートは完全にシーケンシャルなソートであるため，GPU で実装できない．そこで GPU などのストリームプロセッサに適したほぼ整列したリストに有効なブロックトランジションソートを用いる．

6.2.1　コヒレンシを利用したソート

ブロックトランジションソートは奇遇トランジションソートの考え方を一般化したソートである．奇遇トランジションソートはではまず偶数番目の要素が上の要素と比較され，次に奇数番目の要素が上の要素と比較される．この 2 種類の操作を繰り返すことによってソートが行われる．このソートは n 個の要素が逆の順番に並んでいたときに最も効率が悪く，$O(n)$ の操作を行わなければならない．しかしほぼ整列したリストに対しては数回の操作でソートが完了する．たとえば 2 つの要素が逆転している場合は 1 回もしくは 2 回の操作でソートが完了する．

ブロックトランジションソートでは 1 回の操作で 2 つの要素の比較を行うのではなく，全体のリストを $n/2$ 個の要素からなるブロックに分解して，隣接した 2 個のブロックの n 個の要素をソートする．1 回目の操作では偶数番目のブロックと上のブロックをあわせた n 個の要素がソートされ，2 回目の操作では奇数番目のブロックとその上のブロックをあわせた n 個の要素がソートされる．ブロックトランジションソートでは $n/2$ 個の範囲での要素の逆転は 1 回もしくは 2 回の操作でソートすることができる．このようにほぼ整列したリストにはブロックトランジションソートは有効なソートである．2 個のブロックのソートにはどのソートを用いてもよい．

ここでは各ブロックのソートにバイトニックマージソートを用いた[14]．シェアードメモリを用いることのできないシェーダでの実装では 1 回の要素の交換の後にデバイスメモリに書き出さなければならないが，シェアードメモリを用いることによってデータをすべてシェアードメモリに格納できる要素数ならば，低速なデバイスメモリへの書き込みはソート完了後に 1 度行えばよいため，効率的である．

6.2.2　スライスグリッドへのソートの導入

あるプロセッサの担当している計算領域から出ていった粒子は，それ以降はそのプロセッサで計算する必要がないため，フラグを立てて，物理量の配列の後方に移動される．こうすることによって計算に必要な粒子が小さい粒子番号に固まるようになり，物理的に同時に実行されるスレッドの処理の発散も防ぐことができるようになる．ただしソートすることで粒子番号は変わり，リナンバリングを行っていることになる．

6.2.3　データの送信

既存研究では Y 軸で空間をスライスに分割し，XZ 平面のスライスを作成していたが，この軸は任意にとることができる．計算領域を X 軸方向に複数のプロセッサで分割するとき

© 2008 Information Processing Society of Japan

は，X 軸方向に 2 ボクセル分のデータを隣接したプロセッサに送る必要がある．そのため，スライスグリッドの分割軸を X 軸に定め，YZ 平面のスライスを作成すれば，送信する必要があるデータは連続した 2 枚のスライスのデータとなり，メモリ上で連続した領域のデータを送ればよく，容易に転送するデータを取り出すことができる．具体的にはそれらのスライスの先頭のボクセルの番号を読み出し，そのアドレスからそれらのスライスに存在するボクセルの数のデータを送る．X 軸でスライスに分割するということはそれぞれのボクセル座標で同じ X 座標のもののバウンディングボックスを求め，その内部のみのボクセルをメモリに確保するため，均一格子を用いた場合より，転送データ量を比較すると効率的である．

7. レンダリング

複数の GPU で計算されたデータを 1 つの GPU がレンダリングを行う．レンダリングにはそれぞれの GPU が計算している全粒子の座標をレンダリングを行う GPU に送らなければならない．たとえば 100 万粒子の計算を 4 GPU で行うと，それぞれの GPU は 25 万粒子のデータを送るだけでよいが，レンダリングを行う GPU は 100 万粒子を受け取り，レンダリングしなければならず，レンダリングを行う GPU の負荷が高くなる．本研究で計算しているような数十万から百万までの粒子数だと単にそれぞれの粒子を球として可視化するだけでも負荷が高い．もちろんこれらの計算粒子から流体の表面抽出を行い，流体を半透明にレンダリングすることも考えられるが，この計算負荷はさらに高く，現在研究されている手法を用いてもリアルタイムでは行うことができない[8]．本研究では粒子を球としてレンダリングする場合のみを考える．このような場合には内部の粒子は表面の粒子に隠れているためレンダリングする必要はない．レンダリングの負荷を下げるためにレンダリングを行う GPU が表面を抽出する方法も考えられるが，この GPU は 100 万粒子から表面抽出を行わなければならず，1 GPU へ高い負荷がかかる．そこでレンダリングを行う GPU の負荷を減らすために，本研究ではシミュレーションを行っている GPU がそれぞれ不必要な粒子データを削除し，レンダリングに必要なデータのみを転送した．具体的にはシミュレーションを行っている GPU がそれぞれ粒子の密度を計算し，その密度を用いて表面粒子を抽出する．そしてこの表面粒子のみを転送し，レンダリングを行う GPU が表面だけを描画する．粒子の密度の計算は粒子に働く力を計算するときに用いた粒子番号が格納された格子を用いて計算する．これによってシミュレーションを行う GPU の計算コストは増えるが，転送されるデータ量が減りレンダリングを行う GPU の計算負荷が下がる．レンダリングを行う GPU が表面抽出を行う場合と比較すると，3 つの利点がある．1 つ目は表面抽出の計算の負荷が 1 GPU に集中せず，複数の GPU に分散されることである．またレンダリングを行う GPU が表面抽出を行うには近傍粒子探索を効率化するために，格子に粒子番号を格納しなければならないが，シミュレーションを行う GPU が表面抽出を行うとシミュレーションに使ったデータ，つまり粒子番号が格納された格子を用いることができるので効率が良い．これが 2 つ目の利点である．そして最後にシステム全体を見たときに転送されるデータ量を大幅に削減できるため，データ転送のコストという観点で見たときにも効率的になっている．

8. ベンチマーク

以下での計算はすべて C++，CUDA 1.1 を用いて実装し，Intel Core2 Q6600 の CPU，GeForce 8800GT，Tesla S870 の GPU を搭載した PC 上で実行した結果である．なお Tesla S870 には 4 つの C870 の GPU が搭載されているため，本研究で用いた PC には 5 GPU が搭載されていることになる．

GPU 間で直接データ転送を行うことはできない．そのため本研究ではビデオメモリに格納されているデータを 1 度メインメモリに読み戻してから，別の GPU のビデオメモリに転送を行う．そこでビデオメモリ，メインメモリ間のデータ転送速度のベンチマークを行った．**図 5** にそれぞれの GPU が逐次にデータ転送を行った場合の結果と，すべての GPU にいっせいに並列でデータ転送を行った結果を示す．ここではメインメモリからビデオメモリへの

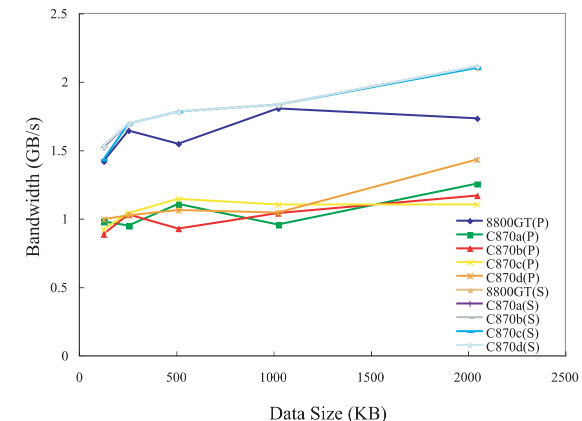

図 5　5 GPU を用いた並列でのデータ転送と逐次でのデータ転送のバンド幅の計測
Fig. 5　Parallel and serial bandwidth test on 5 GPUs.

© 2008 Information Processing Society of Japan

データ転送を測定した．この図から GPU が逐次にデータ転送を行った場合は送るデータサイズが大きくなるほど転送速度が速くなっていることが分かる．さらに，どの GPU でも転送速度はほぼ同じであることが分かる．しかし，並列でデータ転送を行った場合は挙動が異なる．1 GPU が 1 つの PCIExpress のスロットに接続されている GeForce 8800GT は逐次にデータ転送を行った場合に近い結果が得られた，それぞれ 2 つの GPU が 1 つのスロットに接続されている S870 ではそれぞれの GPU の転送速度が逐次にデータ転送を行った場合に比べて大きく下回っている．しかし，この場合 1 個の PCIExpress のスロットでの総データ転送量は 2 つの GPU の転送量の和になるため，単位時間あたりの転送量は 1 GPU の転送量よりも大きなものになっている．また GPU によって，データ転送速度にぶれがあることも分かる．

9. 結果と議論

粒子径を 1 として $256 \times 128 \times 128$ の計算領域内でのシミュレーションを 1，2，4 GPU で行った．4 GPU での計算では計算領域は**図 6** に示すように分割され，それぞれの GPU に計算を割り当てた．また 500,000 粒子の 4 GPU での計算の様子を**図 7** に示す．これらの計算体系において粒子数を変化させて計算時間を計測した結果を**表 1** に示す．2 GPU での計算ではそれぞれの GPU がその接続された領域のデータの転送を行えばよいが，4 GPU では両端の GPU は隣接した GPU に，それ以外の 2 つの GPU は両隣の 2 つの GPU にデータ転送を行わなければならないため，同じ計算では転送量は 2 倍になる．この計算でも同じ粒子数だと 4 GPU では 2 GPU の転送時間のほぼ 2 倍になっている．またこのシミュレーションでは 2 GPU を用いると 1 つの GPU が計算する計算粒子数は約半分になり，4 GPU を用いると 1 GPU が計算する計算粒子数は約 1/4 になる．この表からシミュレーションのみにかかる計算時間は 2 GPU では 1 GPU のときの計算時間の約半分，4 GPU を用いると約 1/4 になっており，計算速度が GPU の数にスケールしていることが分かる．なおこの計算時間にはデータ転送の時間は含まれていないが，複数 GPU で計算するために行わなければならない操作も含まれており，その計算時間が十分小さいことが分かる．しかし，複数 GPU で計算を行うときには 1 GPU での計算と異なり GPU でのデータの管理と転送を行う必要がある．この転送時間も含めた 1 タイムステップにかかる計算時間では，4 GPU を用いた場合は，粒子数が少なくシミュレーション時間が短い場合は転送が占める割合が高くなり 1 GPU を用いた場合の約 2.5 倍の計算速度に，粒子数が多くなると転送が占める割合が低くなり 1 GPU を用いた場合の約 3 倍の計算速度になっている．

仮にサーバクライアント型の計算モデルを用いたとすると，毎タイムステップにおいてまずサーバとなる CPU がそれぞれの粒子を計算するプロセッサを計算し，それぞれのプロセッサに送るデータを用意する必要がある．そしてそのデータを送り，各 GPU に計算させた後にデータをすべて回収し，CPU がメインメモリ上の粒子データを更新する．60 万粒子を用いた計算を考えると毎タイムステップにおいて CPU は全粒子の座標と速度などの粒子の物理量を GPU に送らなければならない．このデータ量は 16.8 MB であり，図 5 のベンチマークの最大値である 1 GB/s でデータ転送ができるとしてもこの転送には 16.8 ms かかる計算になる．さらに計算後に同じデータ量を再度転送しなければならないため，サーバク

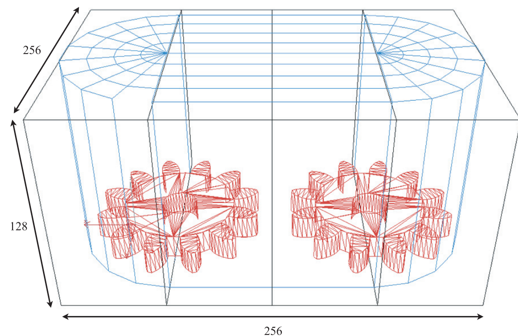

図 6　計算体系．計算領域は 4 つの領域に分割され，1 GPU は 1 つの領域の計算を行う
Fig. 6　Compuation geometry. Computation domain is divided into four domins. A GPU is responsible for a domain.

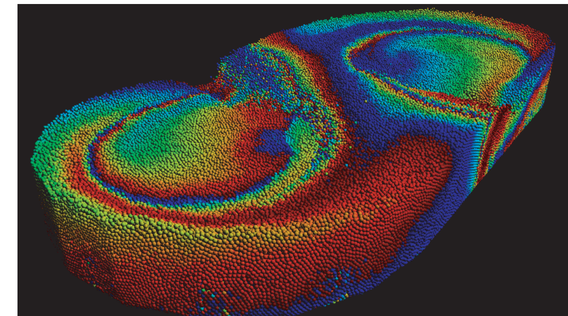

図 7　500,000 粒子のシミュレーション
Fig. 7　A screenshot from a simulation using 500,000 particles.

© 2008 Information Processing Society of Japan

表 1　計算時間の比較（ミリ秒）
Table 1　Comparison of calculation time (in milliseconds).

	1 GPU		2 GPUs		4 GPUs	
N of P	Sim	Total	Sim	Total	Sim	Total
200 K	17.57	17.57	10.56	11.31	6.22	7.06
400 K	33.53	33.53	17.23	18.44	9.68	13.97
600 K	53.82	53.82	24.98	27.09	13.89	18.31
800 K	71.96	71.96	31.17	37.75	19.99	24.49
1 M	93.76	93.76	44.45	46.59	23.19	30.69

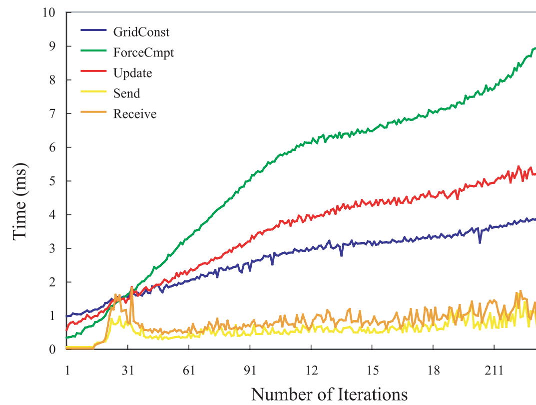

図 8　2 GPU を用いたときの計算時間
Fig. 8　Computation time using 2 GPUs.

ライアント型の計算モデルを採用した場合にデータ転送のみにかかる時間は 33.6 ms である（実際にはゴースト領域の粒子データも転送する必要があるのでより転送時間は増加する）．4 GPU を用いたときに 1 タイムステップの計算にかかる時間が 13.89 ms であるため，計算時間の 2 倍以上の時間をデータ転送のみに費やすことになってしまうため，非常に計算の効率が悪い．さらにこの計算モデルを採用した場合には CPU がデータを管理する時間もかかってしまう．この計算時間の推定から，本研究で行ったような 1 タイムステップの計算時間が非常に短い計算には本手法が適していると考えることができる．

また**図 8** と**図 9** にこの計算体系に粒子を流入させたシミュレーションの 2 GPU と 4 GPU での計算時間の変化を示す．この計算では総粒子数が 500,000 になるまで流入を続けた．2 GPU と 4 GPU での計算時間を比較すると，すべての計算において 4 GPU での計算時間

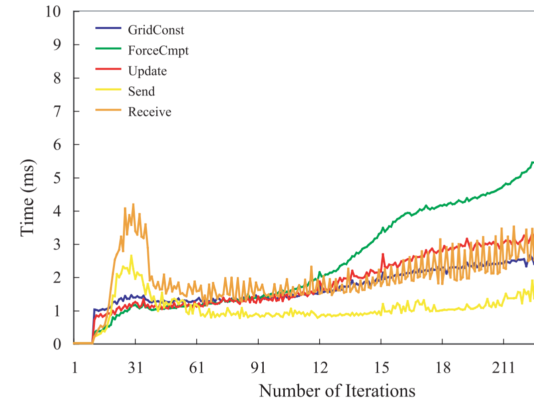

図 9　4 GPU を用いたときの計算時間
Fig. 9　Computation time using 4 GPUs.

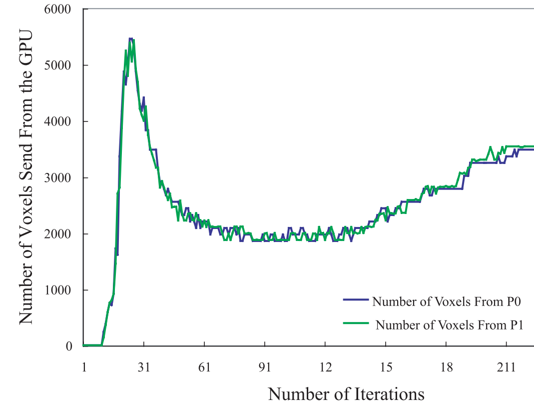

図 10　転送されるボクセル数
Fig. 10　Number of transfered voxels.

は約半分になっていることが分かる．さらに図 8 の 2 GPU の結果を見るとデータの送信と受信にかかる時間に激しいぶれが生じていることが分かる．これはベンチマークでも確認されたように GPU と CPU でのデータ転送にかかる時間が安定していないことによるものであり，さらにこれらは 200 タイムステップ付近ではそれぞれ平均すると 1 ミリ秒程度であり，全体の計算時間に占める割合が十分に小さいことが分かる．それに比べ図 9 の 4 GPU

© 2008 Information Processing Society of Japan

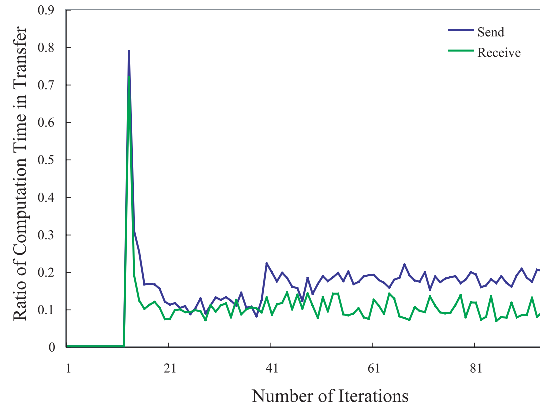

図 11　転送時間におけるデータの処理計算の占める割合
Fig. 11　Ratio of the data processing in transfer time.

表 2　レンダリングのフレームレートの比較（フレーム/秒）．Solid ではすべての粒子がレンダリングされており，surface では表面の粒子のみレンダリングされている
Table 2　Comparison of rendering framerate (frames per second). In solid, all the particles are rendered. In surface, only surface particles are rendered.

N of P	Solid	Surface
10 K	151	181
20 K	123	177
30 K	99	164
40 K	80	160

での結果を見るとデータ転送のふれ幅がさらに大きくなっていることが分かる．これは前述のように 4 GPU での計算の場合，計算領域の両端で接続されている GPU は両隣の GPU にデータ転送する必要があるため，2 GPU のときの処理を 2 回行う必要あるため，ぶれ幅も約 2 倍になっている．転送時間を見ても 2 GPU のときの約 2 倍になっている．また**図 10** にこの 4 GPU を用いたときの計算において転送されるボクセルの数を示す．この図は 0 番目と 3 番目の GPU が 1 番目の GPU に送る量を示している．転送量がタイムステップによって変化しているのは，スライスグリッドを用いて動的に転送するボクセルを計算しているためである．この計算において均一格子を用いて，固定のボクセルを転送すると，32,768 ボクセルのデータを毎タイムステップで転送しなくてはならない．これと比べるとスライスグリッドを用いることにより，転送量をその約 10%程度に削減することができていることが分かる．

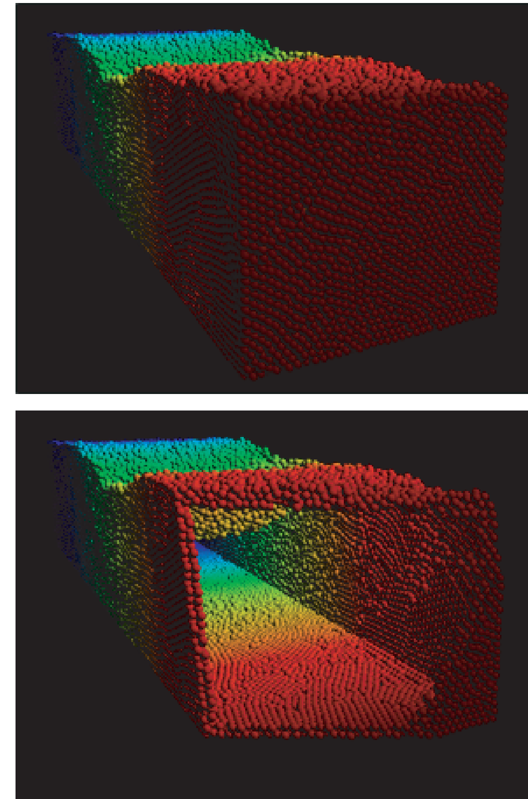

図 12　表面粒子だけのレンダリング．図上は表面粒子のみレンダリングされた結果であり，図下はある面以前の粒子を描画しなかったもの
Fig. 12　Rendering of surface particles. The top figure shows a rendered image and the bottom is an image in which a clipping plane is set to show the cross section.

図 8 と図 9 のデータの送受信の時間はビデオメモリとメインメモリ間の物理的なデータの転送とそのデータの処理を行う時間が含まれている．データを送信するためには粒子番号が格納されている格子を参照し，送信するバッファを用意する必要があり，データを受信したあとはそのデータを粒子データに追加しなければならない．そこで 2 GPU の計算においてデータ処理の時間の転送処理全体に占める割合を示したものを**図 11** に示す．この図から

© 2008 Information Processing Society of Japan

データ送信の準備の割合は 20%以下であり，データ受信後の処理の割合は約 10%であることが分かる．すなわち図 8 や図 9 におけるデータ転送の時間の大半は物理的なデータの転送時間であり，PCIExpress の転送効率が向上すれば本手法を用いて複数 GPU で粒子法シミュレーションを行ったときには転送時間が十分小さくなり，計算時間は GPU の数にほぼ比例するようになる．

最後に**表 2** に全粒子をレンダリングした場合と，表面の粒子のみをレンダリングした場合のフレームレートを示す．なおここでは粒子はポイントスプライトとして描画され，シェーダを用いてライティングの計算を行い球として描画した．表面だけをレンダリングすることで負荷を軽減することができていることが分かる．また**図 12** に表面粒子のみレンダリングされた結果と，その断面を示す．レンダリングの質を落とすことなく，内部の粒子のみ削除されていることが分かる．

10. 結　　論

本研究では複数の GPU を用いて粒子法シミュレーションの 2 段階の並列化を行った．複数の GPU 上で並列化する際に，ピアツーピア型の計算モデルを用い，並列化のオーバヘッドを減らした．また粒子法シミュレーションの計算領域を分割し，1 つの計算領域を 1 つのプロセッサに割り当て，各プロセッサがそれぞれの計算データを管理するモデルを開発した．またプロセッサ間のデータ転送に近傍粒子探索を効率化するために構築した格子を再利用することによって動的なデータ管理のオーバヘッドをおさえ，計算を行う GPU の数にほぼスケールした計算性能を出した．最後に複数の GPU でシミュレーションを行い，1 つの GPU でレンダリングを行うシステムにおいてレンダリングの負荷をシミュレーションを行っている GPU に分散させる手法も提案した．

しかし本手法の欠点としては計算領域を固定し，複数プロセッサ上での計算領域の分割があらかじめ決まっているということである．本手法で最も良い性能を出せるのは粒子が計算領域内に均等に分布しているときであり，粒子が 1 つのプロセッサの計算領域内に固まっている場合は並列化しても性能が向上しない．よって今後の課題としてはどのような粒子配置であっても，計算負荷をすべてのプロセッサ上で均等にすることがあげられる．

本研究は 1PC に複数個の GPU が搭載されている環境で実装と評価を行ったが，本手法はこれ以外の計算機環境にも用いることができる．たとえば，複数台の PC を接続した GPU クラスタにも適用可能である．また GPU を用いない計算環境でも有用である．たとえばマルチコア CPU のみを用いた場合や，一般的な PC クラスタでの計算にも用いることができる．

参　考　文　献

1) Cundall, P.A. and Strack, O.D.L.: A discrete numerical model for granular assemblies, *Geotechnique*, Vol.29, pp.47–65 (1979).
2) Fan, Z., Qiu, F., Kaufman, A. and Yoakum-Stover, S.: Gpu cluster for high performance computing, *Proc. ACM/IEEE Conference on Supercomputing*, p.47 (2004).
3) Harada, T.: *GPU Gems3*, chapter Real-time Rigid Body Simulation on GPUs, Addison-Wesley Pearson Education (2007).
4) Harada, T., Koshizuka, S. and Kawaguchi, Y.: Sliced data structure for particle-based simulations on gpus, *Proc. Graphite*, pp.55–62 (2007).
5) Harada, T., Koshizuka, S. and Kawaguchi, Y.: Smoothed particle hydrodynamics on gpus, *Proc. Computer Graphics International*, pp.63–70 (2007).
6) Harada, T., Tanaka, M., Koshizuka, S. and Kawaguchi, Y.: Acceleration of distinct element method using graphics hardware, *Trans. JSCES*, Vol.20070011 (2007).
7) Harris, M., Segupta, S. and Owens, J.D.: *GPU Gems3*, chapter Parallel Prefix Sum (Scan) with CUDA, Addison-Wesley Pearson Education (2007).
8) Kanamori, Y., Szego, Z. and Nishita, T.: Gpu-based fast ray casting for a large number of metaballs, *Computer Graphics Forum*, Vol.27, No.2, pp.351–360 (2008).
9) Koshizuka, S. and Oka, Y.: Moving-particle semi-implicit method for fragmentation of incompressible fluid, *Nucl. Sci. Eng.*, Vol.123, pp.421–434 (1996).
10) Monaghan, J.J.: Smoothed particle hydrodynamics, *Annu. Rev. Astrophys.*, Vol.30, pp.543–574 (1992).
11) Nakasato, N., Mori, M. and Nomoto, K.: Smoothed particle hydrodynamics with grape and parallel virtual machine, *Astrophysical Journal*, Vol.484, pp.608–617 (1997).
12) NVIDIA: Compute unified device architecture.
http://www.nvidia.com/object/cuda_home.html
13) Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E. and Purcell, T.J.: A survey of general-purpose computation on graphics hardware, *Eurographics 2005, State of the Art Reports*, pp.21–51 (2005).
14) Purcell, T.J., Cammarano, M., Jensen, H.W. and Hanrahan, P.: Photon mapping on programmable graphics hardware, *Proc. ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pp.41–50 (2003).
15) Ribeiro, F.L.B. and Ferreira, I.A.: Parallel implementation of the finite element method using compressed data structures, *Computational Mechanics*, Vol.41, No.1, pp.31–48 (2007).
16) Stratford, K. and Pagonabarraga, I.: Parallel simulation of particle suspensions

© 2008 Information Processing Society of Japan

with the lattice boltzmann method, *Computers & Mathematics with Applications*, Vol.55, No.7, pp.1585–1593 (2008).

17) Thomaszewski, B. and Blochinger, W.: Parallel simulation of cloth on distributed memory architectures, *Proc. Eurographics Symposium on Parallel Graphics and Visualization*, pp.35–42 (2006).

18) Wang, X., Guo, L., Tang, D., Ma, J., Yang, Z. and Li, J.: Parallel implementation of micro scale pseudo particle simulation for particle fluid systems, *Chomputers & Chemical Engineering*, Vol.29, No.7, pp.1543–1553 (2005).

19) 越塚誠一：数値流体力学，培風館 (1997).

20) 越塚誠一：粒子法，丸善 (2005).

21) 越塚誠一，原田隆宏，田中正幸，近藤雅裕：粒子法シミュレーション，培風館 (2008).

22) 原田隆宏，田中正幸，越塚誠一，河口洋一郎：粒子法シミュレーションの並列化，情報処理学会論文誌，Vol.48, No.11, pp.3557–3567 (2007).

23) 入部綱清，藤澤智光，柴田和也，越塚誠一：Mps 法を用いた流体並列解析に関する基礎的研究，*Trans. JSCES*, Vol.20060015 (2006).

(平成 20 年 3 月 12 日受付)
(平成 20 年 9 月 10 日採録)

原田　隆宏（正会員）

昭和 56 年生．平成 18 年東京大学大学院工学系研究科システム量子工学専攻修士課程修了．同年東京大学大学院情報学環助手．平成 19 年 4 月助教．平成 20 年より Havok にて物理シミュレーションの研究開発に従事．

政家　一誠

昭和 56 年生．平成 17 年東京大学大学院工学系研究科システム量子工学専攻修士課程修了．現在，博士課程在籍のかたわら，プロメテック・ソフトウェア株式会社でテクニカルディレクターとして物理シミュレーションの研究開発に従事．

越塚　誠一

昭和 37 年生．昭和 61 年東京大学大学院工学系研究科原子力工学専攻修士課程修了．同年東京大学工学部助手．平成 3 年博士（工学）．同年講師．平成 5 年助教授．平成 16 年教授．連続体の力学シミュレーションの研究に従事．特に粒子法の開発を行う．平成 17 年に丸善より『粒子法』を出版．

河口洋一郎

昭和 27 年生．昭和 51 年九州芸術工科大学（現，九州大学）卒業．昭和 53 年東京教育大学大学院（現，筑波大学大学院）修了．平成 4 年筑波大学芸術学系助教授．平成 10 年東京大学大学院工学系研究科・人工物工学センター教授．平成 12 年東京大学大学院情報学環教授．昭和 50 年から CG に着手し，世界的 CG アーティストとして活躍中．

© 2008 Information Processing Society of Japan